ALGORITMOS Y ESTRUCTURAS DE DATOS
SEMANA Nº 4

OBJETIVOS DE LA CUARTA SEMANA

Clase teórica
- Arreglos simples.

Clase práctica
- Ejercicios de clases con código y diagramas de flujo.

CLASE TEORICA

ARREGLOS

Hasta ahora solo hemos tenido que trabajar con algunas variables en cada uno de los programas que hemos realizado. Sin embargo, en más de una ocasión tendremos que manipular conjuntos mas grandes de valores. Por ejemplo, para calcular la temperatura media del mes de agosto necesitaremos conocer los 31 valores correspondientes a la temperatura media de cada día. En este caso, podríamos utilizar una variable para introducir los 31 valores, uno cada vez, y acumular la suma en otra variable. Pero que ocurrirá con los valores que vayamos introduciendo? que cuando tecleemos el segundo valor, el primero se perdera; cuando tecleemos el tercero, el segundo se perdera, y así sucesivamente. Cuando hayamos introducido todos los valores podremos calcular la media, pero las temperaturas correspondientes a cada día se habrán perdido. ¿Qué podríamos hacer para almacenar todos esos valores? Pues, podríamos utilizar 31 variables diferentes; pero ¿que pasaría si fueran 100 o mas valores los que tuvieramos que registrar? Ademas de ser muy laborioso el definir cada una de las variables, el código se vería enormemente incrementado.

En este semana, aprendera a registrar conjuntos de valores, todos del mismo tipo, en unas estructuras de datos llamadas *arreglos.* Así mismo, aprenderá a registrar cadenas de caracteres, que no son más que conjuntos de caracteres, o bien, silo prefiere, arreglos de caracteres.

Si los arreglos son la forma de registrar conjuntos de valores, todos del mismo tipo (int, float, double, char, String, etc.), que haremos para almacenar un conjunto de valores relacionados entre si, pero de diferentes tipos? Por ejemplo, almacenar los datos relativos a una persona como su *nombre, direccion, telefono,* etc. Ya hemos visto que esto se hace definiendo una clase; en este caso, podria ser la clase de objetos *persona.* Posteriormente podremos crear tambien arreglos de objetos, cuestion que aprenderemos mas adelante.

INTRODUCCION

Un **arreglo** es una **estructura de datos** compuesta por varios elementos, todos del mismo tipo y almacenados consecutivamente en memoria. Cada elemento puede ser accedido directamente por el nombre de la variable arreglo seguido de uno o mas subindices encerrados entre corchetes.

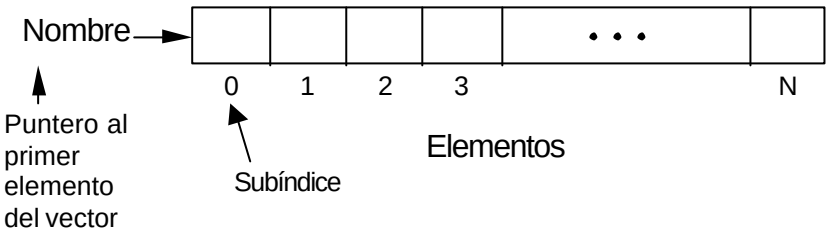
Un arreglo está compuesto de un número de elementos finitos, es de tamaño fijo y tiene elementos homogéneos. Finitos indica que hay un último elemento, tamaño fijo indica que el tamaño del arreglo no cambio una vez creado el mismo y homogéneo significa que todos los elementos son del mismo tipo. En resumen es una estructura de datos lineal, estática y temporal.

El uso de arreglos nos permite manipular un gran número de datos relacionados, sin necesidad de definir una variable para cada uno de ellos. Cada uno de los elementos de un arreglo puede ser cualquiera de los tipos de datos conocidos.

Los arreglos pueden ser de una o varias dimensiones, los de una dimensión se llaman **vectores,** los de dos dimensiones **matrices** y los de mas de dos dimensiones arreglos multidimensionales.

VECTORES

Es un arreglo de una dimensión, se utiliza una sola variable de subindice para acceder a cada elemento del vector.



Para crear y utilizar un vector hay que realizar tres operaciones: declararlo, crearlo e iniciarlo.

Declaración

La declaración de un vector, se hace indistintamente de una de las dos formas siguientes:

tipo[] nombre;
tipo nombre[];

donde *tipo* indica el tipo de los elementos del vector, que pueden ser de cualquier tipo primitivo o referenciado; *y nombre* es un identificador que nombra al vector. Los corchetes modifican la definición normal del identificador pare que sea interpretado por el compilador como un vector.

Las siguientes líneas de código son ejemplos de declaraciones de vectores:

```
int[] m;
float[] temperatura;
COrdenador[] ordenador;    //COrdenador es una clase de objetos.
```

La primera línea declara un vector de elementos de tipo int; la segunda, un vector de elementos de tipo float; y la tercera un vector de objetos *COrdenador.*

Las declaraciones no especifican el tamañoo del vector. El tamaño será especificado cuando se cree el vector, operación que se hará durante la ejecución del programa.

Según se ha podido observar, los corchetes se pueden colocar también después del nombre del vector. Por lo tanto, las declaraciones anteriores pueden escribirse también asi:

```
int m[];
float temperatura[];
COrdenador ordenador[]: // COrdenador es una clase de objetos
```

Creación

Despues de haber declarado el vector, el siguiente paso es crearlo o construirlo. Crear un vector significa reservar la cantidad de memoria necesaria para contener todos sus elementos y asignar al nombre del vector una referencia a ese bloque. Esto puede expresarse genéricamente asi:

nombre = new *tipo[tamano];*

donde *nombre* es el nombre del vector previamente declarado; *tipo* es el tipo de los elementos del vector; *y tamaño* es una expresión entera positiva menor o igual que la precisión de un int, que especifica el número de elementos.

El hecho de utilizar el operador new significa que *Java implementa los vectores como objetos,* por lo tanto seran tratadas como cualquier otro objeto.

Las siguientes líneas de código crean los vectores declarados en el ejemplo anterior:

```
m = new int[10];
temperatura = new float[31];
ordenador = new COrdenador[25];
```

La primera línea crea un vector identificado por *m* con 10 elementos de tipo int; es decir, puede almacenar 10 valores enteros; el primer elemento es *m[0]* (se lee: *m sub-cero),* el segundo *m[1]*, ..., y el ultimo *m[9].* La segunda crea un vector *temperatura* de 31 elementos de tipo float. Y la tercera crea un vector *ordenador* de 25 elementos, cada uno de los cuales puede referenciar a un *objeto COrdenador.* Un vector de objetos es un vector de referencias a dichos objetos.

Es bastante común declarar y crear el vector en una misma línea. Esto puede hacerse asi:

tipo[] ncmbre = new *tipo[tamaño];*
tipo ncmbre[] = new tipo[tamaño];

Las siguientes líneas de código declaran y crean los vectores expuestos en los ejemplos anteriores:

```
int[] m = new int[101;
float[] temperature = new float[31];
COrdenador[] ordenador = new COrdenador[25];
```

Cuando se crea un vector, el tamaño de la misma puede ser también especiticado durante la ejecución a través de una variable a la que se asignará como valor el número de elementos requeridos. Por ejemplo, la última línea de código del ejemplo siguiente crea un vector con el número de elementos especificados por la variable *nElementos:*

```
int nElementos;
System.out.print("Numero de elementos del vector: " );
nElementos = In.readInt();
int[] m = new int[nElementos];
```

Inicialización

Un vector es un objeto; por lo tanto, cuando es creado, sus elementos son automáticamente iniciados, igual que sucede con las variables miembro (atributos) de una clase. Si el vector es numérico, sus elementos son iniciados a 0 y si no es numérica, a un valor análogo al 0; por ejemplo, los caracteres son iniciados al valor '\u0000', un elemento booleano a **false** y las referencias a objetos, a **null**.

Si deseamos iniciar un vector con otros valores diferentes a los predeterminados, podemos hacerlo de la siguiente forma:

```
float[] temperatura = (10.2 F, 12.3F, 3.4F, 14.5F, 15.6F, 16.7F);
```

El ejemplo anterior crea un vector *temperatura* de tipo float con tantos elementos como valores se hayan especifcado entre llaves.

Acceso a los elementos

Para acceder al valor de un elemento de un vector se utiliza el nombre del vector, seguido de un subindice entre corchetes. Esto es, un elemento de un vector no es mas que una variable subindicada; por lo tanto, se puede utilizar exactamente igual que cualquier otra variable. Por ejemplo, en las operaciones que se muestran a continuacion intervienen elementos de un vector:

```
int[] m = new int[100];
int k = 0, a=0;
a = m[1] + m[99];
k=50;
m[k]++;
m[k+1] = m[k];
```

Observe que para referenciar un elemento de un vector se puede emplear como subindice una constante, una variable o una expresión de tipo entero. El subindice especifica la pcsición del elemento dentro de l vector. La primera posición es la 0.

Si se intenta acceder a un elemento con un subindice menor que cero o mayor que el número de elementos del vector menos uno, Java lanzara una excepción de tipo **ArrayIndexOutOfBoundsException**, indicando que el subindice esta fuera de los limites establecidos cuando se creo el vector. Por ejemplo, cuando se ejecute la última línea de código del ejemplo siguiente Java lanzara una excepción, puesto que intenta asignar el valor del elemento de subindice 99 al elemento de subindice 100, que esta fuera del rango 0 a 99 válido.

```
int[] m = new int[l00];
int k = 0, a=0; //  ...
k = 99;
m[k+1] = m[k];
```

¿Como podemos aseguramos de no exceder accidentalmente el final de un vector? Verificando la longitud del vector mediante la variable estática length, que puede ser accedida por cualquier vector. Esta es el unico atributo soportado por los vectores. Por ejemplo:

```
int n = m.length; // número de elementos del vector m
```

Trabajar con vectores

Para practicar la teoria expuesta hasta ahora, vamos a realizar un programa que asigne datos a un vector *m de nElementos* elementos *y,* a continuación, como comprobación del trabajo realizado, muestre el contenido de dicho vector. La solución será similar a la siguiente:

```
Número de elementos de la matriz: 3
Introducir los valores de la matriz.
m[0]= 1
m[1]= 2
m[2]= 3

1 2 3
Fin del proceso.
```

Para ello, en primer lugar definimos la variable *nElementos* para fijar el número de elementos del vector, creamos el vector m con ese número de elementos y definimos el subindice i para acceder a los elementos de dicho vector.

```
int nElementos;
nElementos = In.readIntO;
int[] m = new int[nElementos]; // crear el vector m
int i = 0; // subindice
```

El Paso siguiente es asignar un valor desde el teclado a cada elemento del vector.

```
for (i=0; i < nElementos; i++)
{
      System.out.print("m["+ i + "] =");
      m[i] = In.readInt(),
}
```

Una vez cargado el vector lo visualizamos para comprobar el trabajo realizado.

```
for (i = 0; i < nElementos; i++)
      System.out.print(m[i] + " ");
```

El programa completo se muestra a continuacion:

```
import java.io.*;
import In;
public class Vector
{
      private int[] m;             //declara el vector

      public Vector(int nElementos)
      {
            m = new int[nElementos]; // crea el vector m
      }

      //carga los datos en el vector
      public void setVector()
      {
            int i = 0; // subindice

            for (i=0; i < m.length; i++)   //recorre el vector
            {
                  System.out.print("m["+ i + "] =");
                  m[i] = In.readInt(),
            }
```

```
	}

	//muestra los datos del vector
	public void getVector()
	{
		int i = 0; // subindice

		for (i = 0; i < m.length; i++)
		  System.out.print(m[i] + " ");
	}

	//retorna un string con los datos del vector
	public String toString()
	{
		int i = 0; // subindice
		String v = " " ;

		for (i = 0; i < m.length; i++)
		  v = v + m[i] + " ";

		return v;
	}

}//fin clase vector

import java.io.*;
import In;
import Vector;
public class PrueVector{
	public static void main(String args[]){
		System.out.println("\nNúmero de elementos del vector : ");
		int nElementos = In.readInt();

		Vector vec = new Vector(nElementos);  // Instanciamos el vector

		System.out.println("\n Cargar el vector ");
		vec.setVector();                      //  carga los datos en el vector

		System.out.println("\n Los elementos del vector son : ");
		vec.getVector();                      //  muestra los datos del vector

		System.out.println("\n Otra forma de mostrar ");
		System.out.println(vec.toString());     //  muestra los datos del vector
	}
} // fin clase PrueVector
```

El ejercicio anterior nos enseña como cargar y mostrar un vector. El paso siguiente es aprender a trabajar con los valores almacenados en el vector. Por ejemplo, pensemos en un programa que lea la nota media obtenida por cada alumno de un determinado curso, las almacene en un vector y dé como resultado la nota media del curso.

Igual que hicimos en el programa anterior, en primer lugar crearemos un vector *nota* con un número determinado de elementos solicitado a través del teclado. No se permitirá que este valor sea negativo. En este caso interesa que el vector sea de tipo float para que sus elementos puedan almacenar un valor con decimales. Tambien definiremos un indice i para acceder a los elementos del vector, y una variable *suma* para almacenar la suma total de todas las notas.

```
import java.io.*;
import In;
public class Vector2
{
	private float[] notas;              //declara el vector

	public Vector2(int nElementos)
	{
```

```
            notas = new float[nElementos]; // crea el vector m
        }

        //carga los datos en el vector
        public void setVector()
        {
            for (int i=0; i < notas.length; i++)
            {
                System.out.print("nota media del alumno "+ (i+1) + ": ");
                notas [i] = In.readInt(),
            }
        }

        //muestra los datos del vector
        public void getVector()
        {
            for (int i = 0; i < notas.length; i++)
                System.out.print(notas[i] + " ");
        }


        //retorna la suma las notas medias
        public float getSuma()
        {
            float suma = 0F;

            for (int i = 0; i < notas.length; i++)     //suma todas las notas
                suma += notas[i];

            return suma;
        }

        //retorna el promedio de las notas
        public float getPromedio()
        {
            float promedio;

            promedio = getSuma() / notas.length;

            return promedio;
        }


    }//fin clase vector2

    import java.io.*;
    import In;
    import Vector2;
    public class PrueVector2{
        public static void main(String args[]){
            int nAlumnos;// número de alumnos
            System.out.println("\nIngrese la cantidad de alumnos: ");
            //rutina que valida que no se ingrese un numero de alumnos menor a 1
            do
            {
                nAlumnos = In.readInt();
            }while (nAlumnos < 1);

            Vector2 vec = new Vector2(nAlumnos);  // Instanciamos el vector

            System.out.println("\n Cargar el vector ");
            vec.setVector();                      //  carga los datos en el vector
```

```
                //muestra el promedio de las notas
                System.out.println("El promedio de notas es: "+vec.getPromedio());

                System.out.println("\n Los elementos del vector son : ");
                vec.getVector();                        //  muestra los datos del vector
        }
} // fin clase PrueVector2
```

Vectores asociativos

Cuando el índice de un vector se corresponde con un dato, se dice que el vector es asociativo (por ejemplo, un vector *diasMes[13]* que almacene en el elemento de índice 1 los días del mes 1, en el de índice 2 los días del mes 2 y así sucesivamente; ignoramos el elemento de índice 0). En estos casos, la solución del problema resultará más fácil si utilizamos esa coincidencia. Por ejemplo, vamos a realizar un programa que cuente el número de veces que aparece cada una de las letras de un texto introducido por el teclado y a continuación imprima el resultado. Para hacer el ejemplo sencillo, vamos a suponer que el texto solo contiene letras minúsculas del alfabeto ingles (no hay ni letras acentuadas, ni la *ll,* ni la *ñ).* La solución podría ser de la forma siguiente:

```
Introducir un texto.
Para finalizar presionar [Ctrl][z]
Los arreglos mas utilizados son los unidimensionales y los bidimensionales.
a b c d e f g h i j k l m n o p q r s  t u v w x y z
--------------------------------------------------------------------------
5 1 0 3 5 0 1 0 8 0 0 7 3 5 8 0 0 2 10 1 2 0 0 0 1  1
```

Antes de empezar el problema, vamos a analizar algunas de las operaciones que después utilizaremos en el programa. Por ejemplo, la expresión:

```
'z' - 'a' + 1
```

da como resultado 26. Recuerde que cada caracter tiene asociado un valor entero (codigo ASCII) que es el que utiliza la máquina intemamente para manipularlo. Así por ejemplo la `z' tiene asociado el entero 122, la *`a'* el 97, etc. Según esto, la evaluación de la expresión anterior es: 122 - 97 + 1 = 26.

Por la misma razon, si realizamos las declaraciones,

```
int[] c = new int[256];    //la tabla ASCII tiene 256 caracteres
char car =  'a';           //car tie re asignadc      ente c 3.7
```

la siguiente sentencia asigna *a c[97]* el valor 10,

```
c['a' ] = 10;
```

y esta otra sentencia que se muestra a continuación realiza la misma operación, lógicamente, suponiendo que *car* tiene asignado el caracter `a'.

```
c[car] = 10;
```

Entonces, si leemos un caracter (de la 'a' a la 'z'),

```
car = (char)System.in.readO;
```

y a continuación realizamos la operación,

```
c[car]++;
```

¿qué elemento del vector c se ha incrementado? La respuesta es el de subindice igual al código correspondiente al carácter leído. Hemos hecho coincidir el carácter leído con el subindice del vector. Así cada vez que leamos una 'a' se incrementara el contador c[97] o lo que es lo mismo c['a']; tenemos entonces un contador de 'a'. Análogamente diremos para el resto de los caracteres.

Pero que pasa con los elementos c[0] a c[96]? Según hemos planteado el problema inicial quedarían sin utilizar (el enunciado decía: con que frecuencia aparecen los caracteres de la 'a' a la `z'). Esto, aunque no presenta ningún problema, se puede evitar así:

```
c[car - 'a']++;
```

Para *car* igual a `*a'* se trataría del elemento *c[0] y* para *car* igual a `z' se trataría del elemento c[25]. De esta forma podemos definir un vector de enteros justamente con un número de elementos igual al número de caracteres de la `*a'* a la `*z´* (26 caracteres según la tabla ASCII). El primer elemento será el contador de `a´, el segundo el de `b´, y así sucesivamente.

Un contador es una variable que inicialmente vale cero (suponiendo que la cuenta empieza desde uno) y que después se incrementa en una unidad cada vez que ocurre el suceso que se desea contar.

El programa completo se muestra a continuación.

```
import java.io.*;
import In;
public class VectorAsociativo
{
      private int[] c;              //declara el vector

      public VectorAociativo()
      {
            //Crear el vector c con 'z'-'a'+l elementos.
            //Java inicia los elementos del vector a cero.
            c = new int['z'-'a'+1];
      }

      // Leer los caracteres del texto y contabilizarlos
      public void setVector()
      {
            char car;
            final char eof = (char)-1;
            try
            {
                  // Leer el siguiente caracter del texto y contabilizarlo
                  while ((car = (char)System.in.read() != eof)
                  {
                        //Si el caracter leido esta entre la 'a' y la 'z'
                        //incrementar el contador correspondiente
                        if (car >= 'a' && car <= 'z')
                              c[car - 'a']++;
                  }
            }catch (IOException ignorada) {}
      }

      //muestra los datos del vector
      public void getVector()
      {
            char car;
            for (car = 'a'; car <= 'z'; car++)
                  System.out.print(" " + c[car - 'a']);
      }

}//fin clase VectorAsociativo

import java.io.*;
import In;
import VectorAsociativo;
public class PrueVector3{
      public static void main(String args[]){

            VectorAsociativo vec = new VectorAsociativo ();  // Instanciamos el vector

            System.out.println("Introducir un texto.");
            System.out.printin("Para finalizar presione [Ctrl][z]\n");

            vec.setVector();                        //  carga los datos en el vector
```

```
                // Mostrar la tabla de frecuencias
                System.out.println("\n");
                // Visualizar una cabecera "a b c ..
                for (char car = 'a'; car <= 'z'; car++)
                      System.out.print(" " + car);
                System.out.println("\n ----------------------------------");

                //Visualizar la frecuencia con la que han aparecido los caracteres
                vec.getVector();                        //  muestra los datos del vector
                System.out.println();
          }
    } // fin clase PrueVector3
```

CADENAS DE CARACTERES

Las cadenas de caracteres o string no son un tipo primitivo de datos, pero se pueden manejar a través de la clase **String** o **StringBuffer**, que son dos clases destinadas al manejo de cadenas de caracteres. La diferencia entre ambas es simple. La primera representa las cadenas constantes, mientras que la segunda representa las cadenas modificables.

Las cadenas de caracteres en Java son objetos de la clase **String**. Cada vez que en un programa se utiliza un literal de caracteres, Java crea de forma automática un objeto String con el valor del literal. Por ejemplo, la línea de código siguiente visualiza el literal "Fin del proceso", para lo cual, Java previamente lo convierte a un objeto **String**.

```
System.out.println("Fin del proceso.");
```

Básicamente, una cadena de caracteres se almacena como un vector de elementos de tipo **char**:

```
char[] cadena = new char[10]:
```

Igual que sucedía con los vectores numéricos, un vector de caracteres puede ser iniciado en el momento de su definición. Por ejemplo:

```
char[] cadena = ['a','b','c','d'];
```

Este ejemplo define *cadena* como un vector de caracteres con cuatro elementos *(cadena[0] a cadena[3]) y* asigna al primer elemento el caracter 'a', al segundo el caracter 'b', al tercero el caracter 'c' y al cuarto el caracter 'd'.

Puesto que cada caracter es un entero, el ejemplo anterior podría escribirse también asi:

```
char[] cadena = {197, 98, 99, 100};
```

Cada caracter tiene asociado un entero entre 0 y 65535 (codigo Unicode). Por ejemplo, a la '*a*' le corresponde el valor 97, a la 'b' el valor 98, etc. (recuerde que los primeros 128 códigos Unicode coinciden con los primeros 128 codigos ASCII y ANSI).

Si se crea un vector de caracteres y se le asigna un número de caracteres menor que su tamaño, el resto de los elementos quedan con el valor `\O' con el que fueron iniciados. Por ejemplo:

```
char[] cadena = new char[10];
cadena[0] = 'a'; cadena[1] = 'b'; cadena[2] = 'c';  cadena[3] = 'd';
System.out.println(cadena);
```

La llamada a println permite visualizar la cadena. Se visualizan todos los caracteres hasta finalizar el vector, incluidos los nulos (`\0').

Como ya se expuso al hablar de los vectores numérico, un intento de acceder a un valor de un elemento con un subindice fuera de los límites establecidos al crear el vector, daría lugar a que Java lanzara una excepción durante la ejecución.

Leer y escribir una cadena de caracteres

Una forma de leer un caracter del flujo **in** es utilizando el metodo **read**. Entonces, leer una cadena de caracteres supondrá ejecutar repetidas veces la ejecución de read y almacenar cada caracter leído en la siguiente posición libre de un vector de caracteres. Por ejemplo:

```
char[] cadena = new char[40]; // vector de 40 caracteres
int i = 0, car;
try
{
      System.out.print("Introducir un texto: ");
      while ((car = System.in.read()) != '\r' && i < cadena.length)
      {
            cadena[i] = (char)car;
            i++;
      }
}catch (IOException ignorada) {}
```

El ejemplo anterior define la variable *cadena* como un vector de caracteres de longitud 40. Después establece un bucle para leer los caracteres que se tecleen hasta que se presione la tecla *Entrar.* Cada carácter leído se almacena en la siguiente posición libre del vector *cadena.* Finalmente se escribe el contenido de *cadena,* el número de caracteres almacenados, y la dimensión del vector. Se puede observar que el valor dado por el atributo length no es el número de caracteres almacenado en *cadena,* sino la dimensión del vector.

Observe que el bucle utilizado para leer los caracteres tecleados, podría haberse escrito también así:

```
while ((car = System.in.read()) != '\r' && i < cadena.length)
      cadena[i++] = (char)car;
```

Otra forma de leer una cadena de caracteres, consiste en leer una línea de texto de un flujo de la clase **BufferedReader**, conectado al flujo **in**, se utiliza el metodo **readLine**, y la entrada se almacena en un objeto String. El metodo **readLine** lee hasta encontrar el caracter `\r', `*n*' o los caracteres '\r*n*' introducidos al presionar la tecla *Entrar;* estos caracteres son leídos pero no almacenados, simplemente son interpretados como delimitadores. Por ejemplo:

```
//Definir un flujo de caracteres de entrada: flujoE
inputStreamReader isr = new InputStreamReader(System.in);
BufferedReader flujoE = new BufferedReader(isr);

//Definir una referencia al flujo estandar de salida. flujoS
PrintStream flujoS = System.out;

String cadena; // variable para almacenar una línea de texto
Try
{
      flujoS.print("Introduzca un texto: ");
      cadena = flujoE.readLine() ; // leer una línea de texto
      flujoS.println(cadena); //escribir la linea leida
}catch (IOException ignorada) {}
```

El ejemplo anterior define en primer lugar un flujo de entrada, *flujoE,* del cual se podran leer líneas de texto. Después, define una referencia, *flujoS,* al flujo de salida estandar; esto permitirá utilizar la referencia *flujoS* en lugar de System.out. Finalmente lee una línea de texto introducida a través del teclado. Con esa información, el método readLine crea un objeto y devuelve una referencia al mismo que es almacenada en *cadena.* Finalmente, la llamada a println permite visualizar el objeto String.

Comparando el metodo read con el metodo readLine, se puede observar que este último proporciona una forma más cómoda de leer cadenas de caracteres de un flujo y además, devuelve un objeto String cuyos métodos, como veremos a continuación, hacen muy fácil la manipulación de cadenas.

Un vector de caracteres también puede ser convertido en un objeto String, según se muestra a continuación. Por ejemplo:

```
char[] cadena = new char[40]; //matriz de 40 caracteres
String scadena = new String(cadena);
```

Trabajar con cadenas de caracteres

El siguiente ejemplo lee una cadena de caracteres y a continuacidn visualiza el símbolo y el valor ASCII de cada uno de los caracteres de la cadena. La solución será de la forma:

```
Escriba una cadena de caracteres:
Hola ¿que tal?
Caracter = 'H', codigo ASCII = 72
Caracter = 'o', codigo ASCII = 111
```

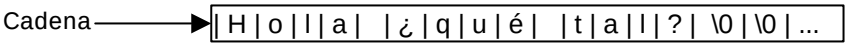
El problema consiste en definir una cadena de caracteres, *cadena, y* asignarle datos desde el teclado utilizando el metodo read. Una vez leída la cadena, se accede a cada uno de sus elementos (no olvide que son elementos de un vector) y por cada uno de ellos se visualiza su contenido y el valor ASCII correspondiente.

Observar que el método println visualiza un elemento de tipo char como un carácter, por lo tanto, para visualizar su valor ASCII es necesario convertirlo explícitamente a int. El programa completo se muestra a continuación.

```
import java.io.*;
import In;

public class Cadena
{
      private char[] cadena;

      public Cadena()
      {
            cadena = new char[80]; // vector de caracteres
      }

      //carga la cadena
      public void setCadena()
      {
            int car, i = 0; // un carácter y el subindice para el vector
            try
            {
                  while ((car=System.in.read()) != '\r' && i < cadena.length)
                        cadena[i++] = (char)car;
            }catch (IOException ignorada) {}
      }

      //muestra la cadena
      public void getCadena()
      {
            int i = 0;
            do
            {
                  System.out.println("Carácter = "+ cadena[i]+ ", código ASCII = "+
                        (int) cadena[i]);
            }while (i<cadena.length && cadena[i] != '\0');
      }
}//fin clase Cadena

import java.io.*;
import In;
import Cadena;
public class PrueCadena{
      public static void main(String args[]){

            Cadena cad = new Cadena();  // Instanciamos la cadena

            System.out.printin("Escriba una cadena de caracteres:
            cad.setCadena();

            System.out.println("\n Los caracteres de la cadena son : ");
            cad.getCadena ();
```

```
        }
    } // fin clase PrueCadena
```

Cuando un usuario ejecute este programa, se le solicitará que introduzca una cadena. Por ejemplo:



Observar que el bucle utilizado para examinar la cadena, para i igual a 0 accede al primer elemento del vector, para i igual a 1 al segundo, y asi hasta llegar al final del vector o hasta encontrar un caracter nulo ('\O') que indica el final de los caracteres tecleados.

En el siguiente ejemplo se trata de escribir un programa que lea una línea de la entrada estándar y la almacene en un vector de caracteres. A continuación, utilizando un método, deseamos convertir los caracteres escritos en minúsculas, a mayúsculas.

En la tabla ASCII los caracteres 'A', ..., `Z', `a', ... , *'z'* estan consecutivos y en orden ascendente de su codigo (valores 65 a 122). Entonces, pasar un caracter de minúsculas a mayúsculas supone restar al valor entero (codigo ASCII) asociado con el caracter, la diferencia entre los códigos de ese caracter en minúscula y el mismo en mayúscula. Por ejemplo, la diferencia 'a'-'A' es 97 - 32 = *65, y* es la misma que `b'-'B', que 'c'-'C', etc.

El programa completo se muestra a continuación.

```
import java.io.*;
import In;

public class Cadena2
{
      private char[] cadena;

      public Cadena2()
      {
            cadena = new char[80]; // vector de caracteres
      }

      //carga la cadena
      public void setCadena()
      {
            int car, i = 0; // un carácter y el subindice para el vector
            try
            {
                  while ((car=System.in.read()) != '\r' && i < cadena.length)
                        cadena[i++] = (char)car;
            }catch (IOException ignorada) {}
      }

      //muestra la cadena
      public void getCadena()
      {
            System.out.println(cadena);
      }


      //convierte las minúsculas a mayúsculas
      public void setMayusculas()
      {
            int i = 0, desp = 'a'-'A';
            for (i=0; i<cadena.length && cadena[i] != '\0'; i++)
                  if (cadena[i] >= 'a' && cadena[i] <= 'z')
                        cadena[i] = (char) (cadena[i] – desp);
      }
}//fin clase Cadena
```

```
import java.io.*;
import In;
import Cadena2;
public class PrueCadena2{
      public static void main(String args[]){

            Cadena2 cad = new Cadena2();  // Instanciamos la cadena

            System.out.printin("Escriba una cadena de caracteres:
            cad.setCadena();

            //tranformar a mayusculas
            cad.setMayusculas();

            cad.getCadena ();
      }
} // fin clase PrueCadena2
```

La solución que se ha dado al problema planteado no contempla los caracteres típicos de nuestra lengua como la ñ o las vocales acentuadas. Este trabajo queda como ejercicio para el lector.

La utilización de vectores de caracteres para la solución de problemas puede ser ampliamente sustituida por objetos de la clase String. La gran cantidad y variedad de métodos aportados por esta clase facilitarán enormemente el trabajo con cadenas de caracteres, puesto que, como ya sabemos, un objeto String encapsula una cadena de caracteres.

Clase String

La clase **String**, que pertenece al paquete java.lang, proporciona métodos para examinar caracteres individuales de una cadena de caracteres, comparar cadenas, buscar y extraer subcadenas, copiar cadenas y convertir cadenas a mayúsculas o a minúsculas. A continuación veremos algunos de los métodos más comunes de la clase **String**. Pero antes sepa que un objeto **String** representa una cadena de caracteres no modificable. Por to tanto, una operación como convertir a mayúsculas no modificará el objeto original sino que devolverá un nuevo objeto con la cadena resultante de esa operación.

Operaciones principales

Las operaciones que se pueden efectuar con un String son las siguientes:

crear la cadena, gracias a **9 (nueve)** constructores diferentes, que permiten construir una cadena vacía, una cadena a partir de otra, una cadena a partir de una serie de octetos(bytes), o incluso a partir de una cadena no constante;

Utilizar cadenas en muy diversas formas, para ello la clase String provee de **48 (cuarenta y ocho)** métodos propios, además de **8 (ocho)** heredados de java.lang.object, que nos posibilitan, por ejemplo:

- Extraer un carácter particular.
- Buscar un carácter o una subcadena en el interior de la cadena.
- Comparar dos cadenas.
- Pasar los caracteres de minúsculas a mayúsculas y al revés.
- Eextraer los blancos.
- Reemplazar caracteres.
- Aañadir otra cadena a la serie de la cadena actual.

La forma fácil de conocer los métodos es: Posiciónese en el help de su entorno de desarrollo Java, y en la búsqueda por contenido digite String y selecciónelo. Obtendrá información a nivel de clase, sus atributos, sus constructores y finalmente sus métodos. El help es un hipertexto, si ud lo recorre y chiquea sobre las palabras clave obtendrá información detallada de cómo usar el método.

Las operaciones de modificación no actúan sobre la propia cadena, sino que crean una nueva cadena.

Por ejemplo:

```
String una = “~MAYUSCULAS";
System.out.println (una.toLowerCase());
```

La línea anterior crea una segunda cadena a partir de la primera, que no sufre cambios. Esta segunda cadena se utiliza para la llamada al método println( ), y después desaparece.

El número de operaciones diferentes que se pueden efectuar sobre cadenas constantes permite reducir el empleo de StringBuffer al mínimo. Sin embargo con esta última clase se puede, por ejemplo:

- Añadir caracteres en el interior o al final de la cadena (no se produce la creación de una nueva cadena, sino que se modifica la cadena actual).
- Añadir directamente a una cadena objetos de tipo int, boolean, etc., sin tener que convertirlos previamente en cadenas de caracteres

Concatenación

La concatenación de cadenas de caracteres consiste en unir una o más cadenas. El lenguaje Java proporciona el operador + para concatenar objetos **String**, así como soporte para convertir otros objetos a objetos **String**. Por ejemplo, en la siguiente línea de código, Java debe convertir las expresiones que aparecen entre paréntesis en objetos **String**, antes de realizar la concatenación.

```
Cadena1+cadena2+cadena3+...
```

Java permite añadir a las cadenas tipos simples, como enteros por ejemplo, sin tener que hacer ninguna conversión de tipos, es decir si se quiere armar una cadena con texto y resultados de algún calculo se lo puede hacer directamente usando el operador +, por ejemplo se quiere mostrar por pantalla un texto y el valor de una variable:

```
short numero = 1;
String descripcion = new string (“número”);
//... cálculo del valor de las variables anteriores
System.out.printín (“Hay “ +numero+” ”+descripcion);
```

visualizará como resultado:

Hay 1 número.

La concatenación de objetos **String** está implementada a través de la clase StringBuffer y la conversión, a través del método **toString** heredado de la clase **Object**.

Tanto la clase como el método citado serán estudiados a continuación. Recuerde que para acceder desde un método de la clase aplicación o de cualquier otra clase a un miembro (atributo o método) de un objeto de otra clase diferente se utiliza la sintaxis *objeto.miembro.* La interpretación que se hace en programación orientada a objetos es que el objeto ha recibido un mensaje, el especificado por el nombre del método, y responde ejecutando ese método.

String(String *valor)*

String es el constructor de la clase **String**. Anteriormente, trabajando con cadenas de caracteres, vimos como utilizar este constructor para crear un objeto **String** a partir de un vector de caracteres. Pero en la mayoría de los casos lo utilizaremos para crear un objeto **String** a partir de un literal o a partir de otro **String**. Por ejemplo, cada una de las líneas siguientes crea un **String**.

```
String strl = “abc”;                    //crea un String “abc”
String str2 = new String(“def”);  // crea un String “def”
String str3 = new String(str1);         // crea un nuevo String “abc”
```

String toString()

Este método devuelve el propio objeto **String** que recibe el mensaje **toString**. Por ejemplo, el siguiente código copia la referencia *str1* en *str2 (*no crea un objeto nuevo referenciado por *str2,* a partir de *str1).* El resultado es que las dos variables, *str1 y str2,* permiten acceder al mismo objeto **String**.

```
String strl = "abc", str2;
str2 = strl.toStringO; // equivale a str2 = strl
```

La misma operación puede ser realizada utilizando la expresi6n *str2 = str1*;

String concat(String *str)*

Este método devuelve como resultado un nuevo objeto **String** resultado de concatenar el **String** especificado a continuación del objeto **String** que recibe el mensaje concat. Por ejemplo, la primera línea de código que se muestra a continuación da como resultado "Ayer llovió" y la segunda "Ayer llovió mucho".

```
System.out.println("Ayer".concat(" llovio"));
System.out.println("Ayer".concat(" llovio".concat(" mucho")));
```

Si alguno de los **String** tiene longitud 0, se concatena una cadena nula. Este otro ejemplo que se muestra a continuación construye un objeto "abcdef" resultado de concatenar *str1 y str2,* y asigna *a str1* la referencia al nuevo objeto.

```
String str1 = "abc", str2 = "def";
Str1 = str1.concat(str2);
```

int compareTo(String *otroString)*

Este método compara lexicográficamente el **String** especificado, con el objeto **String** que recibe el mensaje **compareTo** (el método equals realiza la misma operación). El resultado devuelto es un entero:

< 0 si el **String** que recibe el mensaje es menor que el *otroString,*
= 0 si el **String** que recibe el mensaje es igual que el *otroString y*
> 0 si el **String** que recibe el mensaje es mayor que el *otroString.*

En otras palabras, el método **compareTo** permite saber si una cadena está en orden alfabético antes (es menor) o después (es mayor) que otra y el proceso que sigue es el mismo que nosotros ejercitamos cuando lo hacemos mentalmente, comparar las cadenas caracter a caracter. La comparación se realiza sobre los valores Unicode de cada caracter. El siguiente ejemplo compara dos cadenas y escribe "abcde" porque esta cadena esta antes por orden alfabetico.

```
String str1 = "abcde", str2 = "abcdefg";
if(str1.compareTo(str2) < 0)
      System.out.println(str1);
```

El metodo **compareTo** diferencia las mayúsculas de las minúsculas. Las mayúsculas están antes por orden alfabético. Esto es así porque en la tabla Unicode las mayúsculas tienen asociado un valor entero menor que las minúsculas. El siguiente ejemplo no escribe nada porque "abc" no esta antes por orden alfabético que "Abc".

```
String strl = "abc", str2 = "Abc";
if (strl.compareTo(str2) < 0)
 System.out.println(str1);
```

Si en vez de utilizar el método **compareTo** se utiliza el metodo **compareTolgnoreCase** no se hace diferencia entre mayúsculas y minúsculas.

int length()

Este método devuelve la longitud o número de caracteres Unicode (tipo char) del objeto **String** que recibe el mensaje **length**.

El siguiente ejemplo escribe como resultado: `Longitud: 37`

```
String str1 = "La provincia de Córdoba es muy bonita";
System.out.println ("longitud: "+ str1.length());
```

String toLowerCase()

Este método convierte a minúsculas las letras mayúsculas del objeto **String** que recibe el mensaje **toLowerCase**. El resultado es un nuevo objeto **String** en minúsculas.

String toUpperCase()

Este método convierte a mayúsculas las letras minúsculas del objeto **String** que recibe el mensaje **toUpperCase**. El resultado es un nuevo objeto **String** en mayúsculas.
El siguiente ejemplo almacena en *str1* la cadena *str2* en mayúsculas.

```
String str1, str2 = "Córdoba, la mas bonita ... ";
strl = str2.toUpperCase();
```

String trim()

Este método devuelve un objeto **String** resultado de eliminar los espacios en blanco que pueda haber al principio y al final del objeto **String** que recibe el mensaje **trim**.

boolean startsWith(String *prefijo)*

Este método devuelve un valor true si el *prefijo* especificado coincide con el principio del objeto **String** que recibe el mensaje startsWith.

boolean endsWith(String *sufijo)*

Este método devuelve un valor true si el *sufijo* especificado coincide con el final del objeto **String** que recibe el mensaje endsWith.

String substring(int *IndiceInicial, int IndiceFinal)*

Este método retorna un nuevo **String** que encapsula una subcadena de la cadena almacenada por el objeto **String** que recibe el mensaje **substring**. La subcadena empieza en *IndiceInicial y* se extiende hasta *IndiceFinal - 1,* o hasta el final si *IndiceFinal* no se especifica.

El siguiente ejemplo, elimina los espacios en blanco que haya al principio y al final de *str1,* verifica si *str1* finaliza con "gh" y en caso afirmativo obtiene de *str1* una subcadena *str2* igual *a str1* menos el sufijo "gh".

```
String strl = "abcdefgh", str2 = "";
strl = strl.trimO ;
if (strl.endsWith("gh"))
      str2 = str1.substring(O, str1.length() - "gh".length() );
```

char charAt(int *Indice)*

Este método devuelve el carácter que está en la posición especificada en el objeto **String** que recibe el mensaje **charAt**. El índice del primer carácter es el 0. Por lo tanto, el parámetro *indice* tiene que estar entre los valores 0 y length() - 1, de lo contrario Java lanzará un excepción.

int indexOf(int *car)*

Este método devuelve el índice de la primera ocurrencia del carácter especificado por *car* en el objeto **String** que recibe el mensaje **indexOf**. Si *car* no existe el método **indexOf** devuelve el valor -1. Puede comenzar la búsqueda por el final en lugar de hacerlo por el principio utilizando el método **lastIndexOf**.

int indexOf(String *str)*

Este método devuelve el índice de la primera ocurrencia de la subcadena especificada *por str* en el objeto **String** que recibe el mensaje **indexOf**. Si *str* no existe **indexOf** devuelve -1. Puede comenzar la búsqueda por el final en lugar de hacerlo por el principio utilizando el metodo **lastIndexOf**.

String replace(char *car,* char *nuevoCar)*

Este método devuelve un nuevo **String** resultado de reemplazar todas las ocurrencias *car* por *nuevoCar* en el objeto **String** que recibe el mensaje **replace**. Si el carácter *car* no existiera, entonces se devuelve el objeto **String** original.

static String valueOf(tipo *dato)*

Este método devuelve un nuevo **String** creado a partir del dato pasado como argumento. Puesto que el método es static no necesita ser invocado para un objeto **String**. El argumento puede ser de los tipos boolean, char, char[], int, long, float, double y Object.

```
double pi = Math.PI;
String strl = String.valueOf(pi);
```

Char[] toCharArray()

Este método devuelve un vector de caracteres creado a partir del objeto **String** que recibe el mensaje **toCharArray**.

```
String str = "abcde";
char[] mcar = str.toCharArray();
```

Clase StringBuffer

Del estudio de la clase String sabemos que un objeto de esta clase no es modificable. Se puede observar y comprobar que los metodos que actuan sobre un objeto String con la intención de modificarlo, no lo modifican, sino que devuelven un objeto nuevo con las modificaciones solicitadas. En cambio, un objeto **StringBuffer** es un objeto modificable tanto en contenido como en tamaño.

Algunos de los métodos más interesantes que proporciona la clase **StringBuffer**, perteneciente al paquete java.lang, son los siguientes:

StringBuffer([arg])

Este método permite crear un objeto de la clase **StringBuffer**. El siguiente ejemplo muestra las tres formas posibles de invocar a este metodo:

```
StringBuffer strbl = new StringBuffer();
StringBuffer strb2 = new StringBuffer(80);
StringBuffer strb3 = new StringBuffer("abcde");
System.out.println(strbl + " " + strbl.length()+ " " + strbl.capacity()):
System.out.println(strb2 + " " + strb2.length()+     " + strb2.capacity());
System.out.println(strb3 + " " + strb3.length()+ " " + strb3.capacity());
```

La ejecución de las líneas de código del ejemplo anterior, da lugar a los siguientes resultados:

```
0 16
0 80
abcde 5 21
```

A la vista de los resultados podemos deducir que cuando **StringBuffer** se invoca sin argumentos construye un objeto vacío con una capacidad inicial para 16 caracteres; cuando se invoca con un argumento entero, construye un objeto vacío con la capacidad especificada; y cuando se invoca con un **String** como argumento construye un objeto con la secuencia de caracteres proporcionada por el argumento y una capacidad igual al número de caracteres almacenados más 16.

int length()

Este método devuelve la longitud o número de caracteres Unicode (tipo char) del objeto **StringBuffer** que recibe el mensaje **length**. Esta longitud puede ser modificada por el metodo **setLength** cuando sea necesario.

int capacity()

Este método devuelve la capacidad en caracteres Unicode (tipo char) del objeto **StringBuffer** que recibe el mensaje **capacity**.

StringBuffer append(tipo x)

Este metodo pérmite añadir la cadena de caracteres resultante de convertir el argumento x en un objeto **String**, al final del objeto **StringBuffer** que recibe el mensaje **append**. El *tipo* del argumento *x* puede ser boolean, char, char[], int, long, float, double, String y Object. La longitud del objeto **StringBuffer** se incrementa en la longitud correspondiente al **String** añadido.

StringBuffer insert(int *indice, tipo x)*

Este método permite insertar la cadena de caracteres resultante de convertir el argumento x en un objeto **String**, en el objeto **StringBuffer** que recibe el mensaje **insert**. Los caracteres seran añadidos a partir de la posición especificada por el argumento *indice. El tipo* del argumento *x* puede ser boolean, char, char[], int, long, float, double, String y Object. La longitud del objeto **StringBuffer** se incrementa en la longitud correspondiente al **String** insertado.

El siguiente ejemplo crea un objeto **StringBuffer** con la cadena "Mes de del año", a continuación inserta la cadena "Abril " a partir de la posición 7, y finalmente añade al final la cadena representativa del entero 2002. El resultado sera "Mes de Abril del año 2002".

```
StringBuffer strb = new StringBuffer( "Mes de del año ");
strb.insert("Mes de ".length(), "Abril "); // "Mes de ".length()=7
strb.append(2002);
```

StringBuffer delete(int *p1,* int p2)

Este método elimina los caracteres que hay entre las posiciones *p1 y p2 - 1* del objeto **StringBuffer** que recibe el mensaje delete. El valor *p2* debe ser mayor que *p1.* Si *p1* es igual *que p2,* no se efectuará ningún cambio y si es mayor Java lanzará una excepción.

Partiendo del ejemplo anterior, el siguiente ejemplo elimina la subcadena "Abril " del objeto *strb* yañade en su misma posición la cadena "Mayo ". El resultado será "Mes de Mayo del año 2002".

```
StringBuffer strb = new StringBuffer("Mes de del año ");
strb.insert(7, "Abril ");
strb.append(2002);
strb.delete(7, 13):
strb.insert(7, "Mayo ");
```

StringBuffer replace(int p1, int p2, String str)

Este método reemplaza los caracteres que hay entre las posiciones *p1 y p2* - 1 del objeto **StringBuffer** que recibe el mensaje replace, por los caracteres especificados por *str.* La longitud y la capacidad del objeto resultante serán ajustadas automáticamente al valor requerido. El valor *p2* debe ser mayor que *p1.* Si *p1* es igual que *p2*, la operación se convierte en una inserción, y si es mayor Java lanzara una excepción. Según lo expuesto, el ejemplo anterior, puede escribirse también así:

```
StringBuffer strb = new StringBuffer("Mes de del año ");
strb.insert(7, "Abri ");
strb.append(2002);
strb.replace(7, 13, "Mayo ");
```

StringBuffer reverse()

Este método reemplaza la cadena almacenada en el objeto **StringBuffer** que recibe el mensaje reverse, por la misma cadena pero invertida.

String substring(int *IndiceInicial, int IndiceFinal)*

Este método retoma un nuevo **String** que encapsula una subcadena de la cadena almacenada por el objeto **StringBuffer** que recibe el mensaje substring. La subcadena empieza en *IndiceInicial y* se extiende hasta *IndiceFinal - 1,* o hasta el final si *IndiceFinal* no se especifica.

char charAt(int *indice)*

Este método devuelve el carácter que esta en la posición especificada en el objeto **StringBuffer** que recibe el mensaje **charAt**. El índice del primer carácter es el 0. Por to tanto, el parámetro *indice* tiene que estar entre los valores 0 y length() - 1.

void setCharAt(int *indice, char car)*

Este método reemplaza el caracter que está en la posición especificada en el objeto **StringBuffer** que recibe el mensaje **setCharAt**, por el nuevo caracter especificado. El índice del primer carácter es el 0. Por lo tanto, el parámetro *indice* tiene que estar entre los valores 0 y length() - 1.

String toString()

Este método devuelve como resultado un nuevo **String** copia del objeto **StringBuffer** que recibe el mensaje **toString**.

El siguiente ejemplo copia la cadena almacenada en *strb* en *str.*

```
StringBuffer strb = new StringBuffer("abcde");
String str = strb.toString();
```

Clase StringTokenizer

Esta clase, perteneciente al paquete java.util, permite dividir una cadena de caracteres en una serie de elementos delimitados por unos determinados caracteres. De forma predeterminada los delimitadores son: el espacio en blanco, el tabulador horizontal (\t), el caracter nueva línea (\n), el retorno de carro (\r) y el avance de página (\f).

Un objeto **StringTokenizer** se construye a partir de un objeto **String**. Por ejemplo:

```
StringTokenizer cadena;
cadena = new StringTokenizer(" uno, dos, tres y cuatro");
```

Para obtener los elementos de la cadena separados por los delimitadores, en este caso predeterminados, utilizaremos los metodos **hasMoreTokens** para saber si hay más elementos en la cadena, y **nextToken** para obtener el siguiente elemento. Por ejemplo:

```
while (cadena.hasMoreTokens() )
 System.out.println(cadena.nextToken());
```

Cuando ejecutemos las cuatro líneas de código correspondientes a los dos ejemplos anteriores, el resultado que se mostrará será el siguiente:

```
uno,
dos,
Tres
y
cuatro
```

También se pueden especificar los delimitadores en el instante de construir el objeto **StringTokenizer**. Por ejemplo, la siguiente línea de código especifica como delimitadores la coma:

```
cadena = new StringTokenizer("uno, dos, tres y cuatro", ",");
```

En este caso, et resultado que se obtendrá a partir del objeto *cadena* es el siguiente:

```
uno
dos
tres
y
cuatro
```

La diferencia con respecto a la versión anterior es que ahora no aparece la coma como parte integrante de los elementos, ya que se ha especificado como delimitador y los delimitadores no aparecen. Si queremos que los delimitadores aparezcan como un elemento más, basta especificar true como tercer argumento:

```
cadena = new StringTokenizer("uno, dos, tres y cuatr  o", ",", true);
```

Ahora el resultado será el siguiente:

```
Uno
,
,
dos
,
,
tres
,
y
,
cuatro
```

Conversión de cadenas de caracteres a datos numéricos

Cuando una cadena de caracteres representa un número y se necesita realizar la conversión al valor numérico correspondiente, por ejemplo, para realizar una operación aritmética con el, hay que utilizar los métodos apropiados proporcionados por las clases del paquete java.lang: Byte, Character, Short, Integer, Long, Float, Double y Boolean. Por ejemplo:

```
String strl = "1234";
int datol = Integer.parseInt(strl); // convertir a entero

String str2 = "12.34";
float dato2 = (new Float(str2)).floatValue() ; // convertir a float
```

CLASE PRACTICA

Repaso teorico

Ejercicios prácticos

1. Codificar una clase ArregloNum conteniendo un arreglo de 5 elementos enteros definidos como constantes. Hay que exhibir el arreglo y su acumulado.

```
//VectorNum.java
import Java.io.*;
public class VectorNum{
	private int arreglo []= {20,30,40,30,60};

	public String toString(){		//  Preparando la cadena representativa del objeto
		int acuNum = 0;
		String aux = "\nEl arreglo contiene los siguientes elementos\n";
		aux	+= "Ord.Val\n";
		for(int i =0;i<arreglo.length;i++) {
			aux += " "+i+"  "+arreglo[i]+"\n";
			acuNum+=arreglo[i]; //  Acumulamos
		}
		aux += "\ny su acumulado es: " + acuNum + "\n";
		return aux;		//   Retornamos la cadena
	}
};

//PrueVect.java
import Java.io.*;
import  VectorNum;
public class PrueVect{
	public static void main(String args[]){
		VectorNum vect = new VectorNum(); // Creamos el objeto VectorNum
		System.out.println(vect.toString());
	}
}
```

El arreglo contiene los siguientes elementos

```
Ord.Val
 0  20
 1  30
 2  40
 3  30
 4  60
y su acumulado es: 180
Process Exit...
```

2. Codificar una clase ArregloNum conteniendo un arreglo de n elementos enteros cargados a partir de Math.random(). Hay que exhibir el arreglo y su acumulado. El valor de n es solicitado en el operador en el main() y pasado como argumento al constructor de la clase ArregloNum.

```
//VectorNum.java
import  Java.io.*;
public class VectorNum{
	private int arreglo [];

	public VectorNum(int cant){
		arreglo = new int[cant];
		for(int i = 0; i< arreglo.length;i++)
			arreglo[i] = (int)(100*Math.random());
	}
```

```
        public String toString(){
                int acuNum = 0;
                String aux = "El arreglo contiene los siguientes elementos\n";
                aux       += "Ord.Val\n";
                for(int i =0;i<arreglo.length;i++) {
                        aux += " "+i+"  "+arreglo[i]+"\n";
                        acuNum+=arreglo[i];
                }
                aux += "\ny su acumulado es: " + acuNum + "\n";
                return aux;
        }
};

//PrueVect.java
import  VectorNum;
import In;
public class PrueVect{
        public static void main(String args[]){
                System.out.print("\nCuantos elementos tendra el vector? ");
                int aux = In.readInt();
                VectorNum vect = new VectorNum(aux);    // Creamos el objeto VectorNum
                System.out.println(vect.toString());
        }
}
```

```
Cuantos elementos tendra el vector? 6
El arreglo contiene los siguientes elementos
        Ord.Val
         0  16
         1  89
         2  81
         3  3
         4  34
         5  71
        y su acumulado es: 294
        Process Exit...
```

3. Codificar una clase ArregloNum conteniendo un arreglo de n elementos enteros cargados a partir de entrada de datos por teclado. Hay que exhibir el arreglo e indicar cual es su elemento de menor valor. El valor de n es solicitado en el operador en el main() y pasado como argumento al constructor de la clase ArregloNum.

```
        //VectorNum.java
        import  Java.io.*;
        import In;
        public class VectorNum{
                private int arreglo [];

                public VectorNum(int cant){
                        arreglo = new int[cant];
                        for(int i = 0; i< arreglo.length;i++){
                                System.out.print("\nInforme valor del "+i+" elemento ");
                                arreglo[i] = In.readInt();
                        }
                }

                private int menorElem(){
                        int menor = arreglo[0];int orden = 0;
                        for(int i = 0; i< arreglo.length;i++)
                                if (arreglo[i]<menor){
                                        menor = arreglo[i];
                                        orden = i;
```

```
                }
            return orden;
        }

        public String toString(){
            String aux = "El arreglo contiene los siguientes elementos\n";
            aux       += "Ord.Val\n";
            for(int i =0;i<arreglo.length;i++) {
                aux += " "+i+"  "+arreglo[i]+"\n";
            }
            aux += "\ny su menor elemento es el orden " + menorElem() + "\n";
            return aux;
        }
};

//PrueVectorNum.java
import  VectorNum;
import In;
public class PrueVect{
    public static void main(String args[]){
        System.out.print("\nCuantos elementos tendra el vector? ");
        int aux = In.readInt();
        VectorNum vect = new VectorNum(aux);    // Creamos el objeto VectorNum
        System.out.println(vect.toString());
    }
}
```

```
Cuantos elementos tendra el vector? 5
Informe valor del 0 elemento 12
Informe valor del 1 elemento 15
Informe valor del 2 elemento 18
Informe valor del 3 elemento 6
Informe valor del 4 elemento 20

El arreglo contiene los siguientes elementos
Ord.Val
 0  12
 1  15
 2  18
 3  6
 4  20
y su menor elemento es el orden 3

Process Exit...
```

4. Cuantas veces un carácter está contenido en una cadena?

Para resolverlo usaremos el método **charAt**, para retornar el carácter de la posición **index** y lo comparamos al carácter que estamos buscando. Como la clase String **no tiene** el comportamiento que necesitamos, un ciclo que contabilice la cantidad de veces que ocurre este carácter, definiremos una clase propia **Cadena02** que contenga un objeto String construido a partir de una constante literal recibida desde el main(). Hay otras formas de hacerlo, esta no es mala …

```
//Cadena02.java
import java.lang.String;
public class Cadena02{
    private String cadena;
    private int veces;
    public Cadena02(String hilera){   //     constructor
        cadena = new String(hilera);
    }
    public void contar(char caracter){      //     El ciclo de conteo
        veces = 0;
```

```
                for(int i = 0;i < cadena.length(); i++)
                        if(caracter == cadena.charAt(i)) veces++;
        }
        public void mostrar(char caracter){
                System.out.println(toString() + " el caracter " + caracter + "\n");
        }
        public String toString(){
                String cadAux = new String("\nEn la cadena :\n"
                                        + cadena + " ocurre " + veces + " veces ");
                return cadAux;
        }
}

//PruePruCadena02.java
import  Cadena02;
import In;
public class PruCadena02{
        public static void main(String args[]){
                Cadena02 cadA;
                cadA = new Cadena02 ("Esto es una prueba");
                cadA.contar('a');
                cadA.mostrar('a');
        }
}
```

5. Cuantas veces un carácter está contenido en una cadena y en que posiciones se encuentra?

Es el mismo problema de antes, con el agregado de las posiciones. O sea, necesitamos de más recursos, más comportamiento en los métodos. Para guardar las posiciones nos viene bien un objeto de la clase StringBuffer, que, a diferencia de la clase String, puede ser modificada después de su instanciación. En esa StringBuffer podemos ir agregando las sucesivas posiciones donde el carácter buscado ocurre, separadas por comas, para mayor claridad... manos a la obra. Además, para facilitar la codificación decidimos incluir el carácter buscado como atributo privado del objeto cadena03 .

```
import java.lang.String;
import java.lang.StringBuffer;
public class Cadena03{
        private String cadena;
        private StringBuffer posic;
        private char car;
        private int veces;
        public Cadena03(String hilera, char car){      //     constructor
                cadena = new String(hilera);      //     cadena estatica
                posic  = new StringBuffer();      //     cadena dinamica
                this.car = car;
                veces = 0;
        }

        public void contar(){      //     El ciclo de conteo
                int i;
                for(i = 0;i < cadena.length(); i++)
                        if (car == cadena.charAt(i))
                        {
                                veces++;
                                posic.append(" " + i + ",");
                        }
        }

        public String toString(){
                String cadAux = new String("\n En la cadena: " + cadena + " el caracter "+
                                Car + " ocurre " + veces + " veces \n"  + "en las siguientes
                                posiciones : " + posic);
                return cadAux;
        }
```

```
}


//PruCadena03.java
import  Cadena03;
import In;
public class PruCadena03{
       public static void main(String args[]){
              Cadena03 cadA;
              cadA = new Cadena03 ("Esto es una prueba",`a´);
              cadA.contar();
              System.out.println(cadA.toString());
       }
}
```

6. Cuantas veces una clave de búsqueda (La clave es una cadena que se supone es menor que la cadena en donde buscamos) está contenida en una cadena y en que posiciones se encuentra?

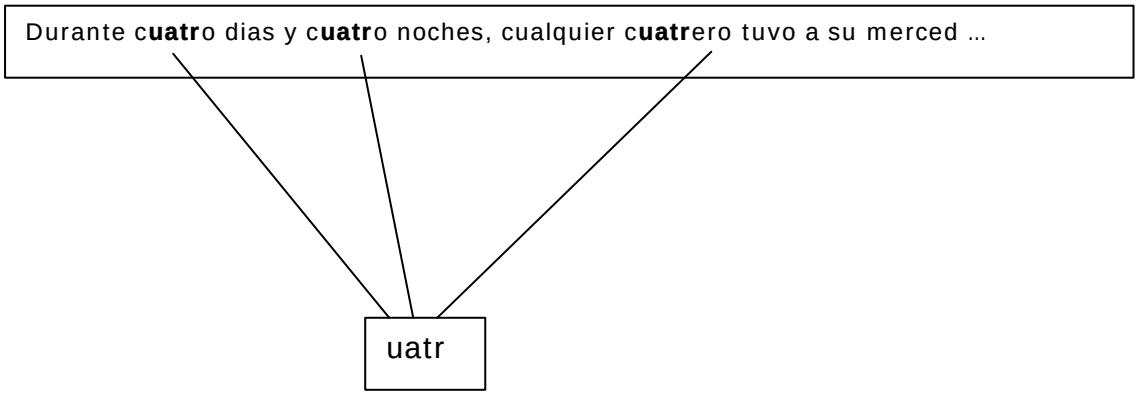
Es un problema parecido al anterior 2. Las diferencias son:

- Buscamos una clave (Conjunto de varios caracteres) en lugar de un solo carácter
- La cadena puede contener espacios separando palabras. La leeremos usando el método readLine de la clase In.
- La clave es una secuencia continua, sin espacios. Puede contener letras, dígitos. La leeremos usando el método readString de la clase In.

Viendo que métodos contiene la clase String que nos puedan servir, encontramos:
public int indexOf(String str): Retorna el índice de la primer ocurrencia de la clave dentro de la cadena.
public int indexOf(String str, int fromIndex): Retorna el índice de la primer ocurrencia de la clave dentro de la cadena a partir de fromIndex

El primero es útil si el enunciado fuera del tipo: Existe una clave en una cadena determinada?. Muy simple, lo podemos resolver en un main() que usa objetos String.
Pero si lo que queremos saber es cuantas veces existe la clave en la cadena y en que posiciones ello ocurre, tenemos que usar una clase propia, con los métodos adecuados. Podemos usar la clase Cadena03 como punto de partida.

Sea la cadena, por ejemplo:



Gráficamente vemos que la clave está contenida tres veces. Dejaremos que el programa nos diga las posiciones.

```
import java.lang.String;
import java.lang.StringBuffer;
import In;
public class Cadena04{
       private String cadena, clave;
       private StringBuffer posic;
       private int veces;
       public Cadena04(String cad, String clav){                    //     constructor
              cadena = cad;
              clave  = clav;
              posic  = new StringBuffer();      //     cadena dinamica para posiciones
```

```
                veces  = 0;
        }
        public void contar(){      //     El ciclo de conteo
                int i = 0,j;
                while(i < cadena.length()){
                        j =  cadena.indexOf(clave,i);
                        if (j >= 0){        //     Encontramos, entonces
                                veces++;            //     contabilizamos,
                                posic.append(" " + j + ",");      //  registramos ocurrencia,
                                i = j;              //     posicionamos indice
                        }
                        i++;                        //     avanzamos indice
                }
        }

        public String toString(){
                String cadAux = new String("\nEn la cadena : " + cadena + "\nla clave " +
                                           clave + " ocurre " + veces + " veces \n"
                                           + "en las siguientes posiciones : " + posic);
                return cadAux;
        }
}

//PrueCadena04.java
import  Cadena04;
import java.io.*;
import In;
public class PruCadena04{
        public static void main(String args[]){
                String cadena, clave;
                System.out.println("\nInforme la cadena a ser analizada, por favor ...");
                cadena = In.readLine();
                System.out.println("\nAhora la  clave de busqueda ...");
                clave  = In.readString();  //     clave de busqueda
                Cadena04 cadA  = new Cadena04 (cadena, clave);
                cadA.contar();
                System.out.println(cadA.toString());
        }
}
```

7. Hacer un programa que realice la Detección de Palíndromos. UD sabe lo que es un Palíndromo? No? Pero si sabe que es capicúa. verdad? Es lo mismo.

```
import java.lang.String;
import java.lang.StringBuffer;
import java.io.*;
import In;
public class Cadena05{
        private String cadena;
        boolean palind;
        public Cadena05(String cad){                    //     constructor
                cadena = cad;                   //     cadena estatica
                palind  = true;                 //     Presuncion a priori
        }

        public void palindromo(){  //     El ciclo de conteo
                int i = 0,j = cadena.length()-1;
                while(i < cadena.length()/2){
                        if (cadena.charAt(i) != cadena.charAt(i)){ //  No es palindromo
                                palind = false;
                                break;
                        }
                        i++;          //     avanzamos
                        j--;                  //     retrocedemos
```

```
                }
        }

        public String toString(){
                String cadAux = new String("\nEl metodo palindromo analiza la cadena: \n"
                                                + cadena
                                                + "\nEs ella palindroma? " + palind);
                return cadAux;
        }
}

import Cadena05;
class PreuCadena05{
        public static void main (String argv []) {
                String cad
                System.out.println("\nInforme la cadena a ser analizada, por favor ...");
                cad = In.readLine();       //     cadena estatica
                Cadena05 cadA = new Cadena05(cad);
                cadA.palindromo();                      //     analizamos
                System.out.println(cadA.toString());
        }
}
```